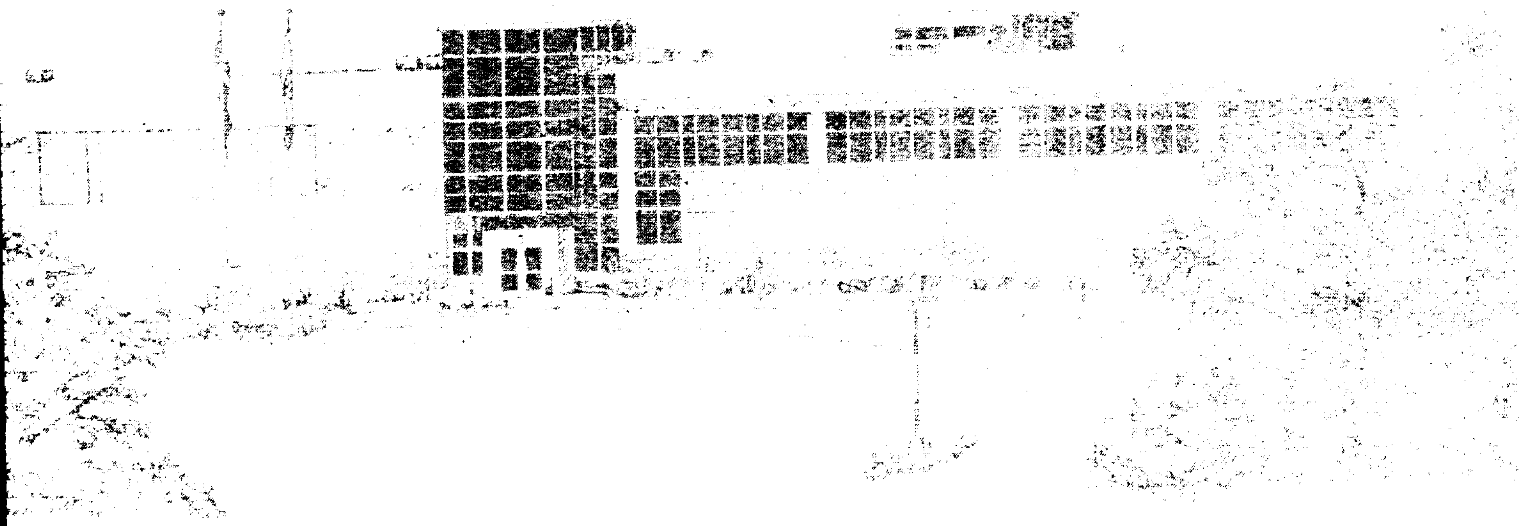
NASA-CR-200925

NASA/WVU Software IV & V Facility
Software Research Laboratory
Technical Report Series

NASA-IVV-95-010
WVU-SRL-95-010
WVU-SCS-TR-95-30
CERC-TR-RN-95-013

Reliable Multicast Protocol Specifications Flow Control and NACK Policy

by John R. Callahan, Todd L. Montgomery, and Brian Whetten



National Aeronautics and Space Administration

West Virginia University

Todd Montgomery
Brian Whetten
John R. Callahan
5 October 1995

The Reliable Multicast Protocol Specification
Flow Control and NACK Policy

RMP Library Version: 1.3b (1.3 Beta)

This appendix presents the flow and congestion control schemes recommended for RMP and a NACK policy based on the whiteboard tool.

Flow and Congestion Control

Because RMP uses a primarily NACK based error detection scheme, there is no direct feedback path through which receivers can signal losses through low buffer space or congestion. Reliable multicast protocols also suffer from the fact that throughput for a multicast group must be divided among the members of the group. This division is usually very dynamic in nature and therefore does not lend itself well to a priori determination. These facts have lead the flow and congestion control schemes of RMP to be made completely orthoganol to the protocol specification. This allows several differing schemes to be used in different environments to produce the best results. As a default, a modified sliding window scheme based on previous algorithms are suggested and described below.

Flow control and congestion control are treated as exactly the same problem in this modified sliding window scheme. A sliding window flow control scheme is an adaptive mechanism that attempts to maintain a constant window of packets in transit. Packets in transit are packets that have been sent, but have not been acknowledged yet. Ideally this window corresponds to the current level of available resources. Other predictive flow control schemes have been proposed and are currently under investigation. These schemes are more applicable to high latency long fat networks, such as ATM. These networks require that hundreds of packets be in transit at once, and the consequences of trying to adaptively size the transmission window when congestion occurs is much too high.

Some very good work has been done in providing efficient congestion control for TCP by Van Jacobson. It is this work that RMP has partially adopted and expanded upon for its flow and congestion control mechanisms. The main four adopted points of the TCP work are:

o Round-Trip-Time Variance Estimation

- o Slow Start
- o Dynamic Window Sizing on Congestion
- o Exponential Retransmit Timer Backoff

Round-trip-time (RTT) of a message is the time it requires for a packet to be sent and a corresponding acknowledgment to arrive at the sender. Round-trip-time variance estimations provide a means of determining how large timeout periods should be on retransmissions based on the average measured length of the round-trip-time and the deviation in the time. It has been observed that when network paths become congested, the variance on packet latency becomes very high compared to it average. "If the network is running at 75% capacity...one would expect the round-trip-time to vary by a factor of 16". It is hoped that by continually estimating the variance and adjusting the average, that an accurate timeout period can be calculated that will virtually eliminate all spurious retransmissions. The elimination of spurious retransmissions allows more bandwidth and processing time to be dedicated to actual useful work, as well as reducing the probability of a false failure detection. The calculation of the timeout period can be effectively done using the following formulas:

$$
\begin{aligned}
Err &= M - A \\
A &= A + g_A(Err) \\
D &= D + g_D(|Err| - D) \\
rto &= A + 4D
\end{aligned}
$$

The g_A and g_D terms are gain terms. M is the round-trip-time measurement. A is the round-trip-time average. D is the round-trip-time mean deviation. And rto is the next timeout period length. Experimentation has shown that 0.0625 and 0.125 are good values for g_A and g_D, respectively.

The slow start algorithm is used to increase the window size from its initial size of one packet to the maximum window size that does not cause congestion. The window size is measured in Minimum Transfer Units, or MTUs. 1 MTU represents a set number of bytes of data in transit. The value of 1 MTU is configurable based on network properties. The slow start algorithm starts by initializing the allowable maximum window size to be 1 MTU. Each time an ACK is received for a packet the window size is incremented 1 MTU. It may not be obvious, but this increases the window size exponentially. The window size will increase from 1 to W on a latency L network path in Lw start actually increases the window size fairly rapidly. Once a sign of congestion occurs, then the window must be reduced. After this first reduction, slow start is not used. But a congestion avoidance scheme is used. This scheme increases the window in a more linear fashion to hopefully avoid congestion. This is done by incrementing the window size by 1/(Window Size) each time an ACK arrives for a packet. A window size of W will therefore only generate at most W ACKs, and an increase of 1/W will increase the window by 1 in one round-trip-time. This increases the window size linearly. In this way, resource limits are probed, but not overrun too quickly.

Under the observations that most lost packets are the result of congestion and not errors and that retransmissions must signal lost packets, then any retransmissions, or expired timers for retransmissions, signal congestion. Congestion must decrease the maximum window size. RMP decreases its window size by 50 each time congestion is encountered. After this decrease the window increases using the linear increase presented above.

Each time a timer expires and a retransmission is needed, the exponential retransmit timer backoff scheme doubles the timer. Once an ACK is received for the packet, however, the value is set to the rto value as calculated. This scheme is applied to all packets that require positive acknowledgments. The timer value must be clamped at a certain maximum value, however, currently 2 seconds. This scheme attempts to ensure that false alarms occur very rarely and that alarms signalling retransmission themselves should not cause even more congestion.

Flow control is addressed by allowing NACKs to also signal dropped packets. Sites that are overrun by senders will drop one or more packets, and will have to send NACKs for those packets. A default NACK control policy is to multicast the NACK to the entire group. Thus the sender will see that its packet was dropped and can reduce its window size exactly the same way it would in congestion control, by 50%. Care must be taken not to perform this decrease multiple times for the same packet, however.

NACK Policy

RMP uses a modified SRM Request/Repair policy (as is used in the wb tool). The goals of any RMP NACK policy should be: (1) reduce the number of NACKs sent to the group to the bare minimum required for any random group topology, (2) the delay between request and repair should be as low as possible, and (3) reduce the number of repairs (responses) to the bare minimum required to repair the inconsistent group members across a random group topology. The SRM Request/Repair policy strives towards these goals.

The basic mechanisms are these: when detecting loss, schedule a NACK Requet timer for a random time in the future. When the timer expires, send a NACK for that timestamp. When receiving the repair, stop retransmission of the NACK. While waiting for the NACK Request Timer to expire, any received NACK for the same timestamp should result in the NACK Request timer being exponentially backed off. When receiving a NACK for a timestamp that the site has, then a NACK Repair timer is scheduled for a random time in the future. When the timer expires, send a repair for the timestamp in the NACK. While waiting for the timer to expire, if a dulicate comes in for that timestamp, then cancel the NACK Repair timer. After receiving a duplicate for a timestamp, or receiving a repair for a timestamp, then ignore any NACKs for that timestamp for a slight period of time. The ranges over which the random times are chosen are incredibly important. For a detailed discussion and analysis, see the SRM paper. Random timer ranges can be based on the round-trip-time of Data packets as kept by flow control.

To implement the SRM policy in RMP requires that the NACK policy be integrated into the OrderingQ abstraction of RMP. Each slot in the OrderingQ must have an associated disposition of any pending NACK request or repair for that slot. The five NACK dispositions are: (0) slot has no pending NACK Request or NACK Repair timers, (1) slot has pending NACK Request timer, (2) slot has pending NACK Repair timer, (3) slot has expired NACK Request timer and is now sending a NACK for timestamp of slot, and (4) slot has expired NACK Repair timer and is ignoring all NACKs for slot.

RMP is an event driven system, so the NACK policy also must be event driven. In this vein, the NACK policy is defined in terms of how the NACK disposition of a slot changes in response to different events.

Event: Detect missing slot in OrderingQ

Action: Set slot state to Packet Requested. Set slot NACK disposition to 1. Schedule NACK Request timer.

Event: Receive NACK

Action: If slot NACK disposition is 0, then set slot NACK disposition to 2, schedule NACK Repair timer. If slot NACK disposition is 1, then exponentially back off NACK Request timer for slot. If slot NACK disposition is 2 or 4, then do nothing.

Event: Receive duplicate for slot

Action: If slot NACK disposition is 2, then remove pending NACK Repair timer for slot, set slot NACK disposition to 4. Schedule Ignore NACK timer for slot. If slot NACK disposition is not 2, then do nothing.

Event: Receive repair for requested slot

Action: If slot NACK disposition is 1, then remove pending NACK Request timer, set slot NACK disposition to 4, schedule Ignore NACK timer. If slot NACK disposition is 3, stop retransmit cycle on NACK, set slot NACK disposition to 4, schedule Ignore NACK timer. In both cases, set slot state to Packet Received.

Event: Expired NACK Request timer

Action: start retransmit cycle for NACK, set slot NACK disposition to 3.

Event: Expired NACK Repair timer

Action: retransmit packet for slot, set slot NACK disposition to 4, schedule Ignore NACK timer.

Event: Expired Ignore NACK timer

Action: set slot NACK disposition to 0.

Event: DeQueuing slot from OrderingQ

```
Action:
  slot NACK disposition = 0: no action
  slot NACK disposition = 1: error (slot not repaired)
  slot NACK disposition = 2: remove NACK Repair timer
  slot NACK disposition = 3: error (slot not repaired)
  slot NACK disposition = 4: remove Ignore NACK timer
```

Ignore NACK timers and pending NACK Repair timers can be ignored when the OrderingQ slot is deQueued because the site is certain that the requesting site has already been repaired if the slot is to be deQueued. This form of message stability notification allows some of the cases where a NACK was prematurely sent to be caught and treated accordingly.

Some future examinations include exmaining what adaptive constant schemes provide RMP with good delay on NACK Request and Repair timers. Another possibility is adjusting timer values based on what kind of packet a slot "probably" contains. A common case for this NACK policy is that a missing ACK is actually the only missing packet and that the corresponding Data or Non-Member Data (NMD) packets for it are in the DataQ. In this case, the OrderingQ is actually missing all the ACK and Data/NMD slots, but only requires the ACK to fill all the slots. By taking a guess at the slots probable contents and adjusting the timer constants accordingly, an ACK could be requested before the NACK Request timer(s) for the Data/NMD packets expires, thus causing them to be removed and not serviced.

Authors' Addresses:

Todd Montgomery
West Virginia University
Morgantown, WV
Email: tmont@cerc.wvu.edu

Brian Whetten
University of California Berkeley
Berkeley, CA
Email: whetten@tenet.cs.berkeley.edu

John R. Callahan
West Virginia University
Morgantown, WV
Email: callahan@cerc.wvu.edu